Part 3

Storing data in the cloud

There is one guy called Singleton in your office who knows all about your file server. If Singleton is out of office, no one else can maintain the file server. As you can imagine, while Singleton is on holiday, the file server crashes. No one else knows where the backup is located, but the boss needs the document now or the company will lose a lot of money. If Singleton had stored his knowledge in a database, coworkers could look up the information. But because the knowledge and Singleton are tidily coupled, the information is unavailable.

Imagine a virtual machine where important files are located on hard disk. As long as the virtual machine is up and running, everything is fine. But everything fails all the time, including virtual machines. If a user uploads a document on your website, where is it stored? Chances are high that the document is persisted to hard disk on the virtual machine. Let's imagine that the document was uploaded to your website but persisted as an object in an independent object store. If the virtual machine fails, the document will still be available. If you need two virtual machines to handle the load on your website, they both have access to that document because it is not tightly coupled with a single virtual machine. If you separate your state from your virtual machine, you will be able to become fault-tolerant and elastic. Let highly specialized solutions like object stores and databases persist your state.

AWS offers many ways to store your data. The following table helps to decide which service to use for your data on a high level. The comparison is only a rough overview. We recommend that you choose 2–3 services that best fit your use case and then jump into the details by reading the chapters to make your decision.

Overview of data storage services

Service	Access	Maximum storage volume	Latency	Storage cost
S3	AWS API (SDKs, CLI), third party tools	unlimited	high	very low
Glacier	S3, AWS API (SDKs, CLI), third party tools	unlimited	extreme high	extreme low
EBS (SSD)	Attached to an EC2 instance via network	16 TB	low	low
EC2 Instance Store (SSD)	Attached to an EC2 instance directly	15 TB	very low	very low
EFS	NFSv4.1, for example from EC2 instance or on-premises	unlimited	medium	medium
RDS (MySQL, SSD)	SQL	6 TB	medium	low
ElastiCache	Redis / memcached protocol	6.5 TB	low	high
DynamoDB	AWS API (SDKs, CLI)	unlimited	medium	medium

Chapter 8 will introduce S3, a service offering object storage. You will learn how to integrate the object storage into your applications to implement a stateless server.

Chapter 9 is about block-level storage for virtual machines offered by AWS. You will learn how to operate legacy software on block-level storage.

Chapter 10 covers highly available block-level storage that can be shared across multiple virtual machines offered by AWS.

Chapter 11 introduces RDS, a service that offers managed relational database systems like PostgreSQL, MySQL, Oracle, or Microsoft SQL server. If your applications use such a relational database system, this is an easy way to implement a stateless server architecture.

Chapter 12 introduces ElastiCache, a service that offers managed in-memory database systems like Redis or Memcached. If your applications need to cache data, you can use an in-memory database to externalize ephemeral state.

Chapter 13 will introduce DynamoDB, a service that offers a NoSQL database. You can integrate this NoSQL database into your applications to implement a stateless server.

8 Storing your objects: S3 and Glacier

This chapter covers

- Transferring files to S3 using the terminal
- Integrating S3 into your applications with SDKs
- Hosting a static website with S3
- Diving into the internals of the S3 object store

Storing data comes with two challenges: ever-increasing volumes of data and ensuring durability. Solving the challenges is hard or even impossible if using disks connected to a single machine. For this reason, this chapter covers a revolutionary approach: a distributed data store consisting of a large number of machines connected over a network. This way, you can store near-unlimited amounts of data by adding additional machines to the distributed data store. And since your data is always stored on more than one machine, you reduce the risk of losing that data dramatically.

You will learn about how to store images, videos, documents, executables, or any other kind of data on Amazon S3 in this chapter. Amazon S3 is a simple-to-use, fully managed distributed data store provided by AWS. Data is managed as objects, so

the storage system is called an *object store*. We will show you how to use S3 to back up your data, how to integrate S3 into your own application for storing user-generated content, as well as how to host static websites on S3.

On top of that, we will introduce Amazon Glacier, a backup and archiving store. On one hand, storing data on Amazon Glacier costs less than storing data on Amazon S3. Conversely, retrieving data from Amazon Glacier takes up to 5 hours, compared to immediate access from Amazon S3.

Examples are 100% covered by Free Tier

The examples in this chapter are completely covered by the Free Tier. As long as you don't run the examples longer than a few days, you won't pay anything. Keep in mind that this only applies if you created a fresh AWS account for this book and nothing else is going on in your AWS account. Try to complete the chapter within a few days; you'll clean up your account at the end.

8.1 What is an object store?

Back in the old days, data was managed in a hierarchy consisting of folders and files. The file was the representation of the data. In an *object store*, data is stored as objects. Each object consists of a globally unique identifier, some metadata, and the data itself, as figure 8.1 illustrates. An object's *globally unique identifier* (GUID) is also known as its *key*; you can address the object from different devices and machines in a distributed system using the GUID.

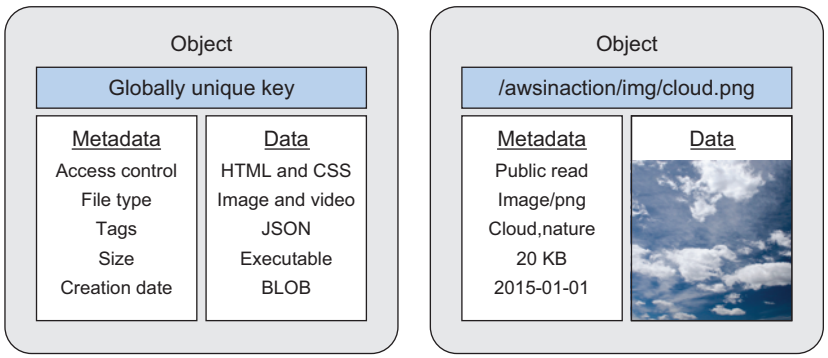

Figure 8.1 Objects stored in an object store have three parts: a unique ID, metadata describing the content, and the content itself (such as an image).

You can use metadata to enrich an object with additional information. Typical examples for object metadata are:

- Date of last modification
- Object size
- Object's owner
- Object's content type

It is possible to request only an object's metadata without requesting the data itself. This is useful if you want to list objects and their metadata before accessing a specific object's data.

8.2 Amazon S3

Amazon S3 is a distributed data store, and one of the oldest services provided by AWS. *Amazon S3* is an acronym for *Amazon Simple Storage Service*. It's a typical web service that lets you store and retrieve data organized as objects via an API reachable over HTTPS.

Here are some typical use cases:

- Storing and delivering static website content. For example, our blog cloudonaut.io is hosted on S3.
- Backing up data. For example, Andreas backs up his photo library from his computer to S3 using the AWS CLI.
- Storing structured data for analytics, also called a *data lake*. For example, I use S3 to store JSON files containing the results of performance benchmarks.
- Storing and delivering user-generated content. For example, I built a web application—with the help of the AWS SDK—that stores user uploads on S3.

Amazon S3 offers unlimited storage space, and stores your data in a highly available and durable way. You can store any kind of data, such as images, documents, and binaries, as long as the size of a single object doesn't exceed 5 TB. You have to pay for every GB you store in S3, and you also incur costs for every request and for all transferred data. As figure 8.2 shows, you can access S3 via the internet using HTTPS to upload and download objects. To access S3 you can use the Management Console, the CLI, SDKs, or third-party tools.

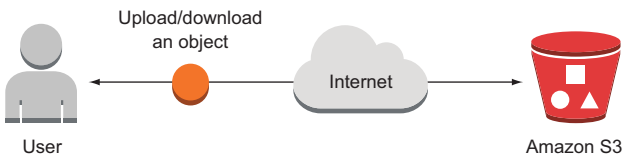

Figure 8.2 Uploading and downloading an object to S3 via HTTPS

S3 uses *buckets* to group objects. A bucket is a container for objects. You can create multiple buckets, each of which has a globally unique name, to separate data for different scenarios. By *unique* we really mean unique—you have to choose a bucket name that isn't used by any other AWS customer in any other region. Figure 8.3 shows the concept.

You will learn how to upload and download data to S3 using the AWS CLI next.

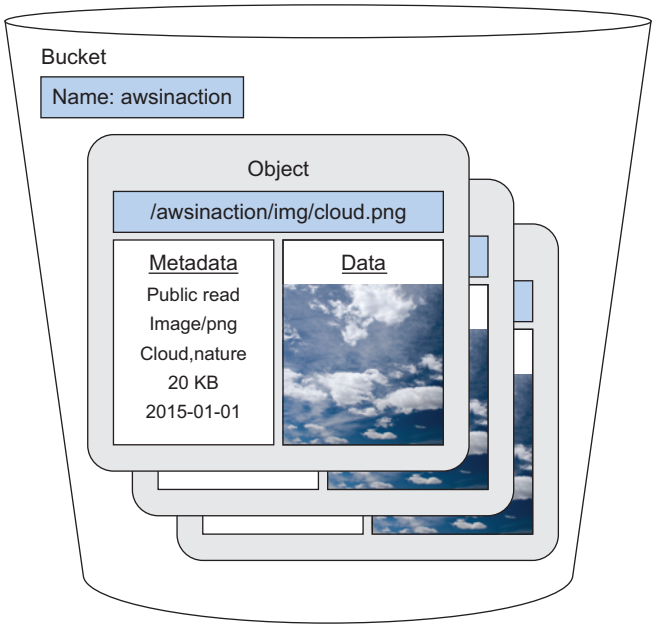

Figure 8.3 S3 uses buckets with a globally unique name to group objects.

8.3 Backing up your data on S3 with AWS CLI

Critical data needs to be backed up to avoid loss. Backing up data at an off-site location decreases the risk of losing data even during extreme conditions like natural disaster. But where should you store your backups? S3 allows you to store any data in the form of objects. The AWS object store is a perfect fit for your backup, allowing you to choose a location for your data as well as storing any amount of data with a pay-per-use pricing model.

In this section, you'll learn how to use the AWS CLI to upload data to and download data from S3. This approach isn't limited to off-site backups; you can use it in many other scenarios as well:

- Sharing files with your coworkers or partners, especially when working from different locations.
- Storing and retrieving artifacts needed to provision your virtual machines (such as application binaries, libraries, configuration files, and so on).
- Outsourcing storage capacity to lighten the burden on local storage systems—in particular, for data that is accessed infrequently.

First you need to create a bucket for your data on S3. As we mentioned earlier, the name of the bucket must be unique among all other S3 buckets, even those in other regions and those of other AWS customers. To find a unique bucket name, it's useful to use a prefix or suffix that includes your company's name or your own name. Run the following command in the terminal, replacing *$yourname* with your name:

```
$ aws s3 mb s3://awsinaction-$yourname
```

Your command should look similar to this one.

```
$ aws s3 mb s3://awsinaction-awittig
```

If your bucket name conflicts with an existing bucket, you'll get an error like this one:

```
[... ] An error occurred (BucketAlreadyExists) [...]
```

In this case, you'll need to use a different value for `$yourname`.

Everything is ready to upload your data. Choose a folder you'd like to back up, such as your Desktop folder. Try to choose a folder with a total size less than 1 GB, to avoid long waiting times and exceeding the Free Tier. The following command uploads the data from your local folder to your S3 bucket. Replace *`$Path`* with the path to your folder and *`$yourname`* with your name. `sync` compares your folder with the /backup folder in your S3 bucket and uploads only new or changed files:

```
$ aws s3 sync $Path s3://awsinaction-$yourname/backup
```

Your command should look similar to this one.

```
$ aws s3 sync /Users/andreas/Desktop s3://awsinaction-awittig/backup
```

Depending on the size of your folder and the speed of your internet connection, the upload can take some time.

After uploading your folder to your S3 bucket to back it up, you can test the restore process. Execute the following command in your terminal, replacing *`$Path`* with a folder you'd like to use for the restore (don't use the folder you backed up) and *`$yourname`* with your name. Your Downloads folder would be a good place to test the restore process:

```
$ aws s3 cp --recursive s3://awsinaction-$yourname/backup $Path
```

Your command should look similar to this one:

```
$ aws s3 cp --recursive s3://awsinaction-awittig/backup/ \
➥ /Users/andreas/Downloads/restore
```

Again, depending on the size of your folder and the speed of your internet connection, the download may take a while.

Versioning for objects

By default, S3 versioning is disabled for every bucket. Suppose you use the following steps to upload two objects:

1. Add an object with key *A* and data *1*.
2. Add an object with key *A* and data *2*.

(continued)

If you download (or *get*) the object with key *A*, you'll download data *2*. The old data *1* doesn't exist any more.

You can change this behavior by turning on *versioning* for a bucket. The following command activates versioning for your bucket. Don't forget to replace `$yourname`:

```
$ aws s3api put-bucket-versioning --bucket awsinaction-$yourname \
➥ --versioning-configuration Status=Enabled
```

If you repeat the previous steps, the first version of object *A* consisting of data *1* will be accessible even after you add an object with key *A* and data *2*. The following command retrieves all objects and versions:

```
$ aws s3api list-object-versions --bucket awsinaction-$yourname
```

You can now download all versions of an object.

Versioning can be useful for backup and archiving scenarios. Keep in mind that the size of the bucket you'll have to pay for will grow with every new version.

You no longer need to worry about losing data. S3 is designed for 99.999999999% durability of objects over a year. For instance, when storing 100,000,000,000 objects on S3, you will lose only a single object per year on average.

After you've successfully restored your data from the S3 bucket, it's time to clean up. Execute the following command to remove the S3 bucket containing all the objects from your backup. You'll have to replace `$yourname` with your name to select the right bucket. `rb` removes the bucket; the `force` option deletes every object in the bucket before the bucket itself is deleted:

```
$ aws s3 rb --force s3://awsinaction-$yourname
```

Your command should look similar to this one:

```
$ aws s3 rb --force s3://awsinaction-awittig
```

You're finished—you've uploaded and downloaded files to S3 with the help of the CLI.

Removing bucket causes BucketNotEmpty error

If you turn on versioning for your bucket, removing the bucket will cause a `BucketNotEmpty` error. Use the Management Console to delete the bucket in this case:

1. Open the Management Console with your browser.
2. Go to the S3 service using the main navigation menu.
3. Select your bucket.
4. Press the Delete bucket button and confirm your action in the dialog that opens.

8.4 Archiving objects to optimize costs

You used S3 to back up your data in the previous section. If you want to reduce the cost of backup storage, you should consider another AWS service: *Amazon Glacier*. The price of storing data with Glacier is about a fifth of what you pay to store data with S3. So what's the catch? S3 offers instant retrieval of your data. In contrast, you have to request your data and wait between one minute and twelve hours before your data is available when working with Glacier. Table 8.1 shows the differences between S3 and Glacier.

Table 8.1 Differences between storing data with S3 and Glacier

	S3	Glacier
Storage costs for 1 GB per month in US East (N. Virginia)	0.023 USD	0.004 USD
Costs for inserting data	Low	High
Costs for retrieving data	Low	High
Accessibility	Immediate upon request	One minute to twelve hours after request. Faster retrieval is more expensive.
Durability	Designed for annual durability of 99.999999999%	Designed for annual durability of 99.999999999%

Amazon Glacier is designed for archiving large files that you upload once and download seldom. It is expensive to upload and retrieve a lot of small files, so you should bundle small files into large archives before storing them on Amazon Glacier. You can use Glacier as a standalone service accessible via HTTPS, integrated into your backup solution, or use S3 integration as in the following example.

8.4.1 Creating an S3 bucket for the use with Glacier

In this section, you'll learn how to use Glacier to archive objects that have been stored on S3 to reduce storage costs. As a rule, only move data to Glacier if the chance you'll need to access the data later is low.

For example, suppose you are storing measurement data from temperature sensors on S3. The raw data is uploaded to S3 constantly and processed once a day. After the raw data has been analyzed, results are stored within a database. The raw data on S3 is no longer needed, but should be archived in case you need to re-run the data processing again in the future. Therefore, you move the raw measurement data to Glacier after one day to minimize storage costs.

The following example guides you through storing objects on S3, moving objects to Glacier, and restoring objects from Glacier. As illustrated in figure 8.4, you need to create a new S3 bucket:

1. Open the Management Console at https://console.aws.amazon.com.
2. Move to the S3 service using the main menu.

3 Click the Create button.
4 Type in a unique name for your bucket (such as awsincation-glacier-$yourname).
5 Choose US East (N. Virginia) as the region for the bucket.
6 Click the Next button.
7 Click the Create button on the last page of the wizard.

Figure 8.4 Creating an S3 bucket via the Management Console

8.4.2 *Adding a lifecycle rule to a bucket*

Back to our example: you are storing raw measurement data in an S3 bucket. The raw data has been analyzed. Next, the raw data should be archived on Glacier. To do so,

add a *lifecycle rule* to your bucket. A lifecycle rule can be used to *archive* or *delete* objects after a given number of days. To add a lifecycle rule that moves objects to Glacier, follow these steps, also illustrated in figure 8.5:

1. Select your bucket named *awsincation-glacier-$yourname* from the bucket overview.
2. Switch to the Management tab.
3. Click the Add Lifecycle Rule button.

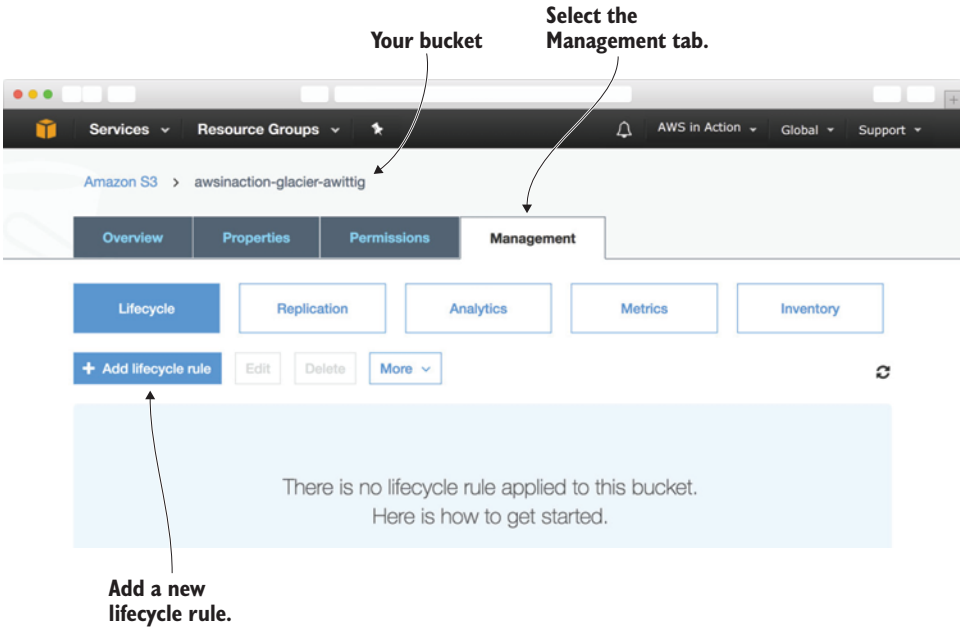

Figure 8.5 Adding a lifecycle rule to move objects to Glacier automatically

A wizard starts that will guide you through the process of creating a new lifecycle rule, as shown in figure 8.6. In the first step of the wizard, you are asked to provide a name and the scope for the lifecycle rule. Type in `glacier` as the rule name, and keep the filter that limits the scope of the rule empty: this applies the rule to all objects within your bucket.

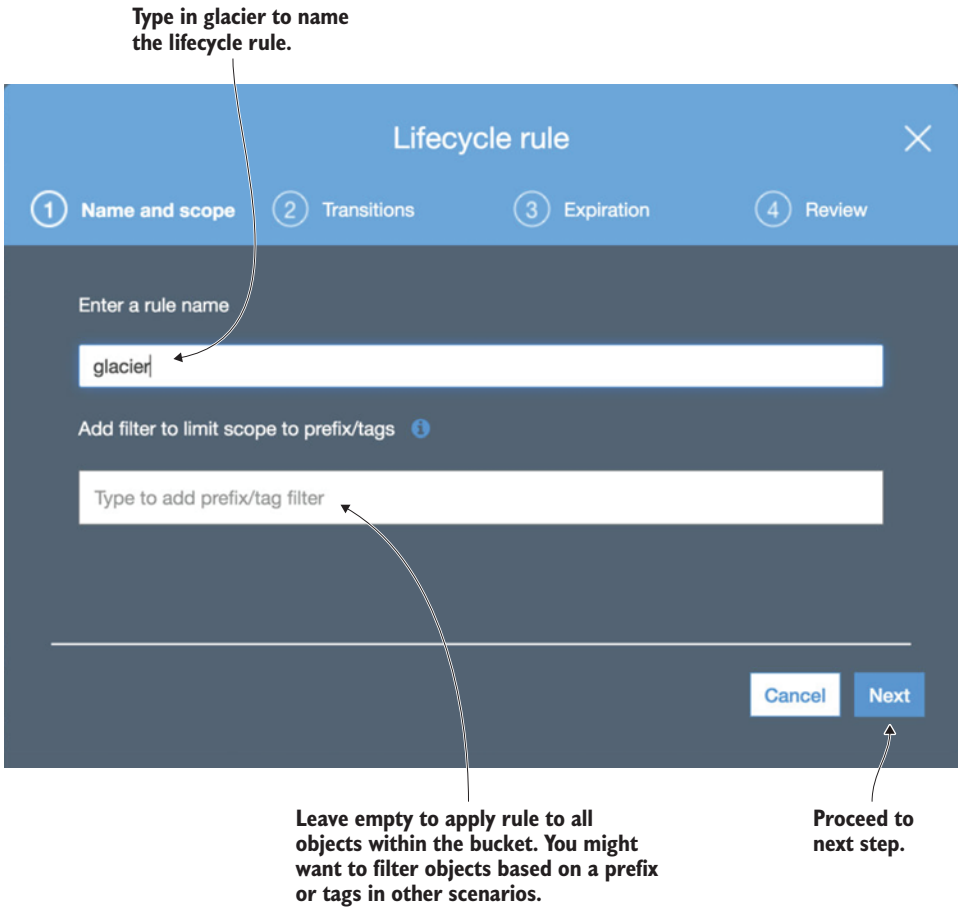

Figure 8.6 Choosing the name and scope of your lifecycle rule

In the next step of the wizard, you will configure the lifecycle rule to archive objects to Glacier. Figure 8.7 shows the details of configuring the transition.

1. Enable transitions for the current version of your objects. As you haven't enabled versioning for your bucket, previous versions of your objects are not available.
2. Select Transition to Amazon Glacier after as the transition type.
3. In the Days After Object Creation field, type in 0 to move your objects to Glacier as quickly as possible.
4. Click Next to proceed to the next step.

Skip the next step of the wizard, which allows you to configure a lifecycle rule to delete objects after a specified period of time. The last step of the wizard shows a summary of your lifecycle rule. Click Save to create your lifecycle rule.

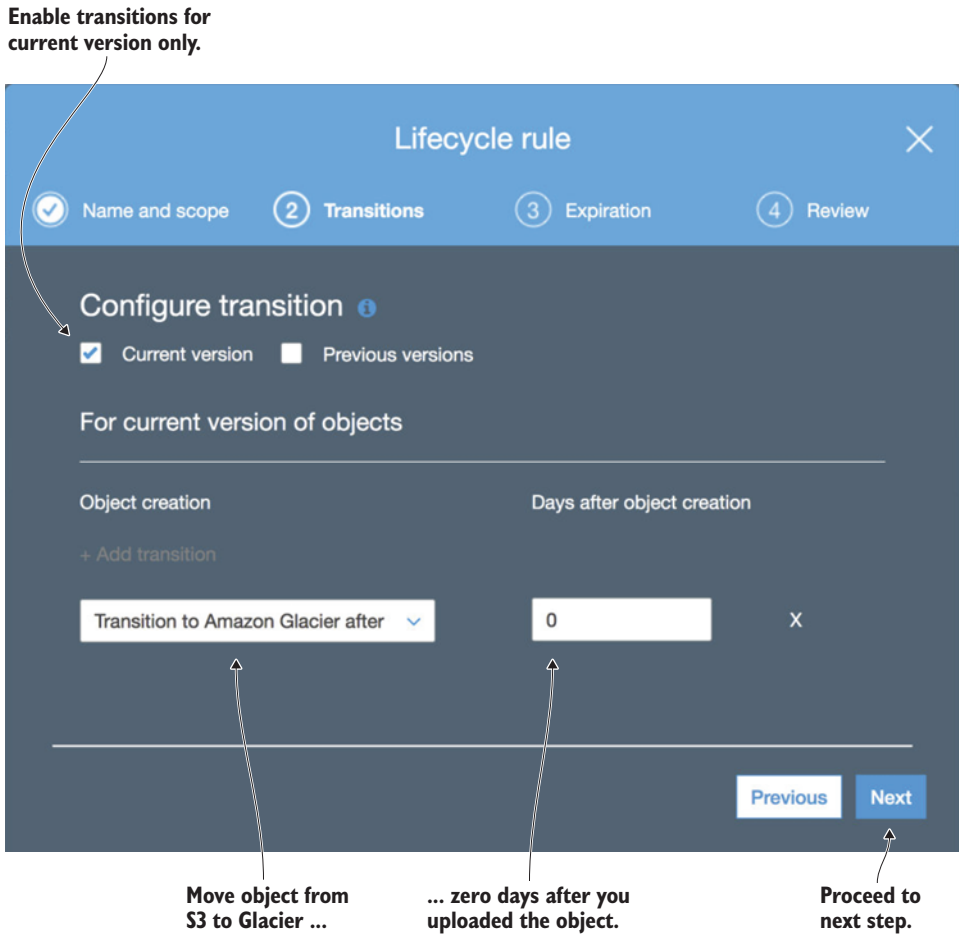

Figure 8.7 Enable transition to Amazon Glacier after 0 days

8.4.3 Experimenting with Glacier and your lifecycle rule

You've successfully created a lifecycle rule that will automatically move all objects from the bucket to Glacier.

> **NOTE** The following example will take more time than usual. It will take up to 24 hours for the lifecycle rule to move your objects to Glacier. The restore process from Glacier to S3 will take 3 to 5 hours.

It's now time to test the process of archiving your raw measurement data. Go back to the overview of your bucket named *awsincation-glacier-$yourname*. Upload a bunch of files by clicking the Upload button. As you probably don't have any files that include measurement data from temperature sensors at hand, feel free to use any kind of data. Your bucket will look similar to what is shown in figure 8.8. By default, all files are stored with storage class Standard, which means they're stored on S3.

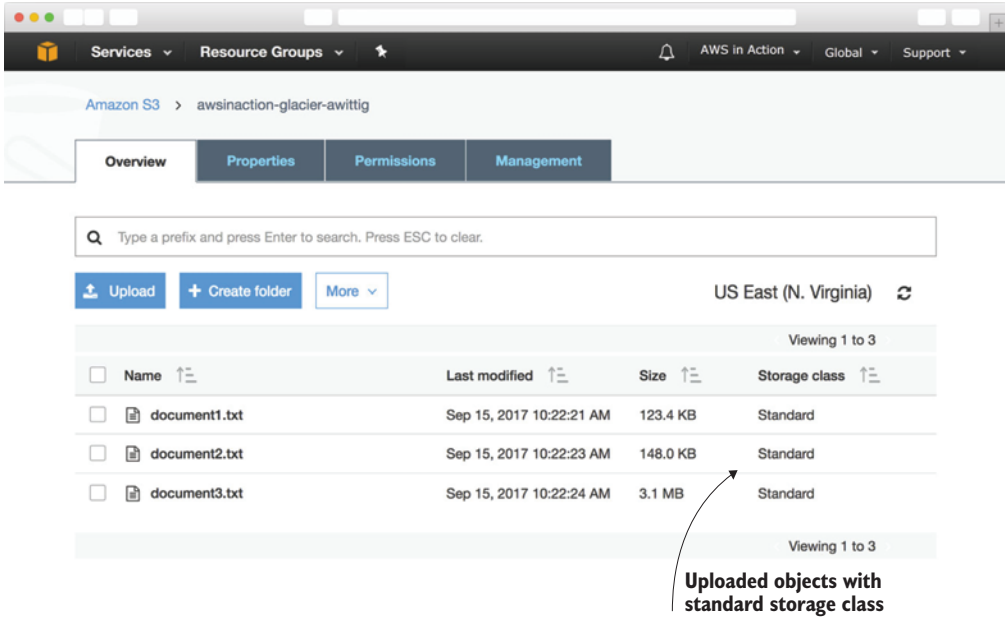

Figure 8.8 Objects with storage class Standard immediately after upload

The lifecycle rule will move the created objects to Glacier. But even though the chosen time gap is 0 days, the move will take up to 24 hours. After your objects have moved to Glacier, the storage class will switch to Glacier, as shown in figure 8.9.

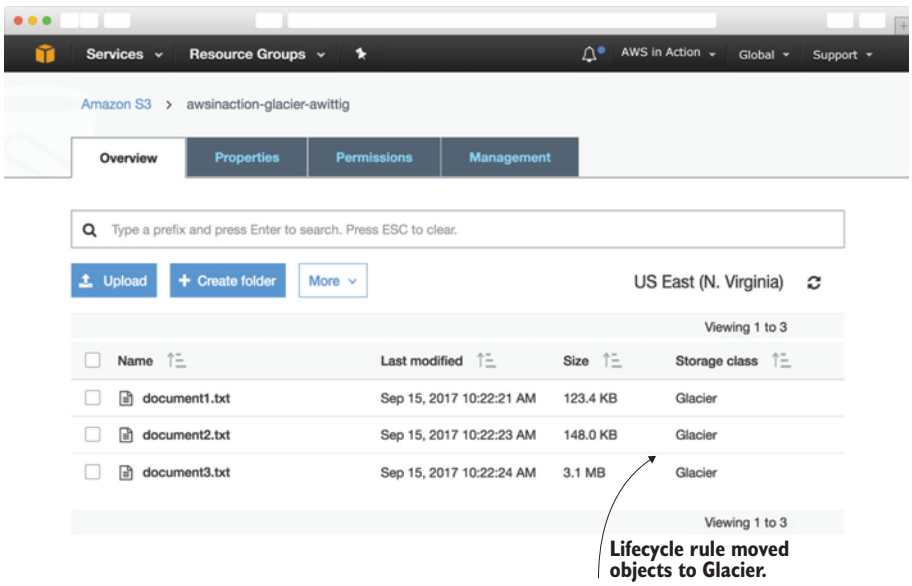

Figure 8.9 Objects have been moved to Glacier after 24 hours.

Let's assume you have found a bug in the processing of measurement data. Incorrect data has been extracted from the raw data and stored in your database. In order to re-run the data analytics, you have to restore the raw data from Glacier.

It is not possible to download files stored in Glacier directly, but you can restore an object from Glacier to S3. Follow the steps illustrated in figure 8.10 to trigger a restore using the Management Console:

1. Open the S3 bucket named *awsincation-glacier-$yourname*.
2. Select the object you want to restore from Glacier to S3.
3. Choose the Initiate Restore action, which is hidden under More.
4. A dialog appears in which you choose how many days the object will be available via S3 after the restore from Glacier, as well as the retrieval option.
5. Click OK to initiate the restore.

Figure 8.10 Restoring an object from Glacier to S3

Retrieval options

Amazon Glacier offers three retrieval options:

- *Expedited*. Data is available within 1–5 minutes; this is the most expensive restore option.
- *Standard*. Data is available within 3–5 hours, for a modest charge.
- *Bulk*. Data is available within 5–12 hours; this is the least expensive restore option.

Restoring an object with the Standard retrieval option usually takes 3 to 5 hours. After the restore is complete, you can download the object. You could re-run your data processing on the raw data now.

Cleaning up

Delete your bucket after you finish the Glacier example. You can do this with the help of the Management Console by following theses steps:

1. Go to the overview of S3 buckets.
2. Select the bucket named *awsincation-glacier-$yourname*.
3. Click the Delete bucket button and confirm the action within the shown dialog.

You've learned how to use S3 with the help of the CLI and the Management Console. We'll show you how to integrate S3 into your applications with the help of SDKs in the next section.

8.5 Storing objects programmatically

S3 is accessible using an API via HTTPS. This enables you to integrate S3 into your applications by making requests to the API programmatically. Doing so allows your applications to benefit from a scalable and highly available data store. AWS offers free SDKs for common programming languages like Go, Java, JavaScript, PHP, Python, Ruby, and .NET. You can execute the following operations using an SDK directly from your application:

- Listing buckets and their objects.
- Creating, removing, updating, and deleting (CRUD) objects and buckets.
- Managing access to objects.

Here are examples of how you can integrate S3 into your application:

- *Allow a user to upload a profile picture.* Store the image in S3, and make it publicly accessible. Integrate the image into your website via HTTPS.
- *Generate monthly reports (such as PDFs), and make them accessible to users.* Create the documents and upload them to S3. If users want to download documents, fetch them from S3.

- *Share data between applications.* You can access documents from different applications. For example, application A can write an object with the latest information about sales, and application B can download the document and analyze the data.

Integrating S3 into an application is one way to implement the concept of a *stateless server*. We'll show you how to integrate S3 into your application by diving into a simple web application called Simple S3 Gallery next. This web application is built on top of Node.js and uses the AWS SDK for JavaScript and Node.js. You can easily transfer what you learn from this example to SDKs for other programming languages; the concepts are the same.

> **Installing and getting started with Node.js**
>
> Node.js is a platform for executing JavaScript in an event-driven environment so you can easily build network applications. To install Node.js, visit https://nodejs.org and download the package that fits your OS. All examples in this book are tested with Node.js 8.
>
> After Node.js is installed, you can verify that everything works by typing `node --version` into your terminal. Your terminal should respond with something similar to `v8.*`. Now you're ready to run JavaScript examples like the Simple S3 Gallery.
>
> Do you want to get started with Node.js? We recommend *Node.js in Action (2nd edition)* from Alex Young, et al. (Manning, 2017), or the video course *Node.js in Motion* from P.J. Evans, (Manning, 2018).

Simple S3 Gallery allows you to upload images to S3 and displays all the images you've already uploaded. Figure 8.11 shows Simple S3 Gallery in action. Let's set up S3 to start your own gallery.

8.5.1 Setting up an S3 bucket

To begin, you need to set up an empty bucket. Execute the following command, replacing *$yourname* with your name or nickname:

```
$ aws s3 mb s3://awsinaction-sdk-$yourname
```

Your bucket is now ready to go. Installing the web application is the next step.

8.5.2 Installing a web application that uses S3

You can find the Simple S3 Gallery application in /chapter08/gallery/ in the book's code folder. Switch to that directory, and run `npm install` in your terminal to install all needed dependencies.

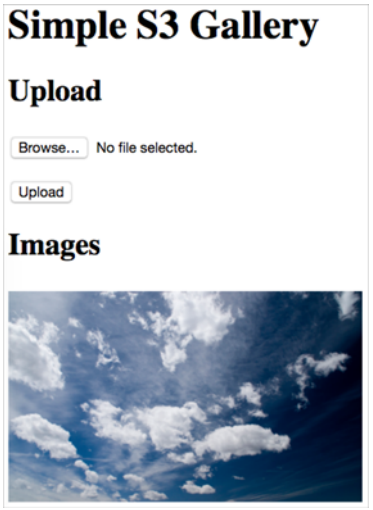

Figure 8.11 The Simple S3 Gallery app lets you upload images to an S3 bucket and then download them from the bucket for display.

To start the web application, run the following command. Replace `$yourname` with your name; the name of the S3 bucket is then passed to the web application:

```
$ node server.js awsinaction-sdk-$yourname
```

> **Where is the code located?**
> You can find all the code in the book's code repository on GitHub: https://github.com/AWSinAction/code2. You can download a snapshot of the repository at https://github.com/AWSinAction/code2/archive/master.zip.

After you start the server, you can open the gallery application. To do so, open http://localhost:8080 with your browser. Try uploading a few images.

8.5.3 Reviewing code access S3 with SDK

You've uploaded images to Simple S3 Gallery and displayed images from S3. Inspecting parts of the code will help you to understand how you can integrate S3 into your own applications. It's not a problem if you don't follow all the details of the programming language (JavaScript) and the Node.js platform; we just want you to get an idea of how to use S3 via SDKs.

UPLOADING AN IMAGE TO S3

You can upload an image to S3 with the SDK's `putObject()` function. Your application will connect to the S3 service and transfer the image via HTTPS. The following listing shows how to do so.

Listing 8.1 Uploading an image with the AWS SDK for S3

```
var AWS = require('aws-sdk');
var uuid = require('uuid');

var s3 = new AWS.S3({'region': 'us-east-1'});

var bucket = [...];

function uploadImage(image, response) {
  var params = {
    Body: image,
    Bucket: bucket,
    Key: uuid.v4(),
    ACL: 'public-read',
    ContentLength: image.byteCount,
    ContentType: image.headers['content-type']
  };
  s3.putObject(params, function(err, data) {
    if (err) {
      console.error(err);
      response.status(500);
```

```
      response.send('Internal server error.');
    } else {
      response.redirect('/');
    }
  });
}
```

Handles success

The AWS SDK takes care of sending all the necessary HTTPS requests to the S3 API in the background.

Listing all the images in the S3 bucket

To display a list of images, the application needs to list all the objects in your bucket. This can be done with the S3 service's `listObjects()` function. The next listing shows the implementation of the corresponding function in the server.js JavaScript file, acting as a web server.

Listing 8.2 Retrieving all the image locations from the S3 bucket

```
var bucket = [...];

function listImages(response) {
  var params = {
    Bucket: bucket
  };
  s3.listObjects(params, function(err, data) {
    if (err) {
      console.error(err);
      response.status(500);
      response.send('Internal server error.');
    } else {
      var stream = mu.compileAndRender(
        'index.html',
        {
          Objects: data.Contents,
          Bucket: bucket
        }
      );
      stream.pipe(response);
    }
  });
}
```

Defines parameters for the list-objects operation

Calls the list-objects operation

The resulting data contains the objects from the bucket list.

Listing the objects returns the names of all the images from the bucket, but the list doesn't include the image content. During the uploading process, the access rights to the images are set to public read. This means anyone can download the images with the bucket name and a random key. The following listing shows an excerpt of the index.html template, which is rendered on request. The `Objects` variable contains all the objects from the bucket.

Listing 8.3 Template to render the data as HTML

Iterates over all objects

Puts together the URL to fetch an image from the bucket

```
[...]
<h2>Images</h2>
{{#Objects}}
  <p><img src=
➥ "https://s3.amazonaws.com/{{Bucket}}/{{Key}}"
➥ width="400px" ></p>
{{/Objects}}
[...]
```

You've now seen the three important parts of the Simple S3 Gallery integration with S3: uploading an image, listing all images, and downloading an image.

Cleaning up

Don't forget to clean up and delete the S3 bucket used in the example. Use the following command, replacing *$yourname* with your name:

```
$ aws s3 rb --force s3://awsinaction-sdk-$yourname
```

You've learned how to use S3 using the AWS SDK for JavaScript and Node.js. Using the AWS SDK for other programming languages is similar.

8.6 Using S3 for static web hosting

We have started our blog cloudonaut.io in May 2015. The most popular blog posts like "5 AWS mistakes you should avoid" (https://cloudonaut.io/5-aws-mistakes-you-should-avoid/), "Integrate SQS and Lambda: serverless architecture for asynchronous workloads" (http://mng.bz/8m5n), and "AWS Security Primer" (https://cloudonaut.io/aws-security-primer/) have been read more than 165,000 times. But we didn't need to operate any VMs to publish our blog posts. Instead we used S3 to host our static website built with a static site generator https://hexo.io. This approach provides a cost effective, scalable, and maintenance-free infrastructure for our blog.

You can host a static website with S3 and deliver static content like HTML, JavaScript, CSS, images (such as PNG and JPG), audio, and videos as well. But keep in mind that you can't execute server-side scripts like PHP or JSP. For example, it's not possible to host WordPress, a CMS based on PHP, on S3.

Increasing speed by using a CDN

Using a content-delivery network (CDN) helps reduce the load time for static web content. A CDN distributes static content like HTML, CSS, and images to nodes all around the world. If a user sends out a request for some static content, the request is answered from the nearest available node with the lowest latency.

> Various providers offer CDNs. Amazon CloudFront is the CDN offered by AWS. When using CloudFront, users connect to CloudFront to access your content, which is which is fetched from S3 or other sources. See the CloudFront documentation at http://mng.bz/Kctu if you want to set this up; we won't cover it in this book.

In addition, S3 offers the following features for hosting a static website:

- *Defining a custom index document and error documents.* For example, you can define index.html as the default index document.
- *Defining redirects for all or specific requests.* For example, you can forward all requests from /img/old.png to /img/new.png.
- *Setting up a custom domain for S3 bucket.* For example, Andreas might want to set up a domain like mybucket.andreaswittig.info for my bucket.

8.6.1 Creating a bucket and uploading a static website

First you need to create a new S3 bucket. To do so, open your terminal and execute the following command, replacing `$BucketName` with your own bucket name. (As we've mentioned, the bucket name has to be globally unique. If you want to redirect your domain name to S3, you must use your entire domain name as the bucket name.)

```
$ aws s3 mb s3://$BucketName
```

The bucket is empty; you'll place an HTML document in it next. We've prepared a placeholder HTML file. Download it to your local machine from the following URL: http://mng.bz/8ZPS. You can now upload the file to S3. Execute the following command to do so, replacing `$PathToPlacerholder` with the path to the HTML file you downloaded in the previous step and `$BucketName` with the name of your bucket:

```
$ aws s3 cp $PathToPlaceholder/helloworld.html \
➥ s3://$BucketName/helloworld.html
```

You've now created a bucket and uploaded an HTML document called helloworld .html. You need to configure the bucket next.

8.6.2 Configuring a bucket for static web hosting

By default, only you, the owner, can access files from your S3 bucket. You want to use S3 to deliver your static website, so you'll need to allow everyone to view or download the documents included in your bucket. A *bucket policy* helps you control access to bucket objects globally. You already know from chapter 6 that policies are defined in JSON and contain one or more statements that either allow or deny specific actions on specific resources. Bucket policies are similar to IAM policies.

Download our bucket policy from the following URL: http://mng.bz/HhgR. You need to edit the bucketpolicy.json file next, as shown in the following listing. Open the file with the editor of your choice, and replace *$BucketName* with the name of your bucket.

Listing 8.4 Bucket policy allowing read-only access to every object in a bucket

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AddPerm",
      "Effect":"Allow",                          <-- Allows access
      "Principal": "*",                          <-- For anyone
      "Action":["s3:GetObject"],                 <-- Read objects
      "Resource":["arn:aws:s3:::$BucketName/*"]  <-- Your bucket
    }
  ]
}
```

You can add a bucket policy to your bucket with the following command. Replace *$BucketName* with the name of your bucket and *$PathToPolicy* with the path to the bucketpolicy.json file:

```
$ aws s3api put-bucket-policy --bucket $BucketName \
➥ --policy file://$PathToPolicy/bucketpolicy.json
```

Every object in the bucket can now be downloaded by anyone. You need to enable and configure the static web-hosting feature of S3 next. To do so, execute the following command, replacing *$BucketName* with the name of your bucket:

```
$ aws s3 website s3://$BucketName --index-document helloworld.html
```

Your bucket is now configured to deliver a static website. The HTML document helloworld.html is used as index page. You'll learn how to access your website next.

8.6.3 Accessing a website hosted on S3

You can now access your static website with a browser. To do so, you need to choose the right endpoint. The endpoints for S3 static web hosting depend on your bucket's region:

```
http://$BucketName.s3-website-$Region.amazonaws.com
```

Replace *$BucketName* with your bucket name and *$Region* with your region. So if your bucket is called `AwesomeBucket` and was created in the default region us-east-1, your bucket name would be:

```
http://AwesomeBucket.s3-website-us-east-1.amazonaws.com
```

Open this URL with your browser, and you should be welcomed by a Hello World website.

Linking a custom domain to an S3 bucket

If you want to avoid hosting static content under a domain like awsinaction.s3-website-us-east-1.amazonaws.com, you can link a custom domain to an S3 bucket, such as `awsinaction.example.com`. All you have to do is to add a CNAME record for your domain, pointing to the bucket's S3 endpoint.

The CNAME record will only work if you comply with the following rules:

- *Your bucket name must match the CNAME record name*. For example, if you want to create a CNAME for `awsinaction.example.com`, your bucket name must be `awsinaction.example.com` as well.
- *CNAME records won't work for the primary domain name (such as `example.com`)*. You need to use a subdomain for CNAMEs like *awsinaction* or *www*, for example. If you want to link a primary domain name to an S3 bucket, you need to use the Route 53 DNS service from AWS.

Linking a custom domain to your S3 bucket only works for HTTP. If you want to use HTTPS (and you probably should), use AWS CloudFront together with S3. AWS CloudFront accepts HTTPS from the client and forwards the request to S3.

Cleaning up

Don't forget to clean up your bucket after you finish the example. To do so, execute the following command, replacing `$BucketName` with the name of your bucket:

```
$ aws s3 rb --force s3://$BucketName
```

8.7 Best practices for using S3

If you're using S3 via the CLI or integrating it into your applications, it's valuable to know about how the object store works. One big difference between S3 and many storage solutions is the fact that S3 is *eventually consistent*, which means you might read stale data after changing an object for a short period of time. Usually you will read the latest version of an object within less than a second after a write, but in rare cases, you might read stale data for much longer. If you don't consider this, you may be surprised if you try to read objects immediately after changing them. Another challenge is designing object keys that offer maximum I/O performance on S3. You'll learn more about both topics next.

8.7.1 Ensuring data consistency

If you're creating, updating, or deleting an object on S3, this operation is *atomic*. This means that if you're reading an object after a create, an update, or a delete, you'll never get corrupted or partial data. But it's possible that a read could return the old data for a while.

If you create, update or delete an object and your request is successful, your change is safely stored. But accessing the changed object immediately might return the *old* version, as shown in figure 8.12. If you retry accessing the object, after a while the new version will be available.

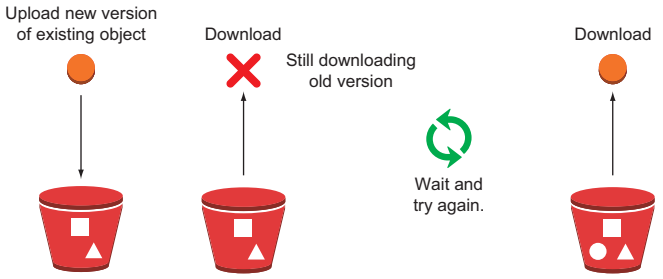

Figure 8.12 Eventual consistency: if you update an object and try to read it, the object may contain the old version. After some time passes, the latest version will be available.

If you don't send a GET or HEAD request to the key of an object before creating the object with a PUT request, S3 offers read-after-write consistency in all regions.

8.7.2 Choosing the right keys

Naming variables and files is difficult. This is especially true for choosing the right keys for objects you want to store in S3. In S3, keys are stored in alphabetical order in an index. The key name determines which partition the key is stored in. If your keys all begin with the same characters, this will limit the I/O performance of your S3 bucket. If your workload will require more than 100 requests per second, you should choose keys for your objects that begin with different characters. As figure 8.13 shows, this will give you maximum I/O performance.

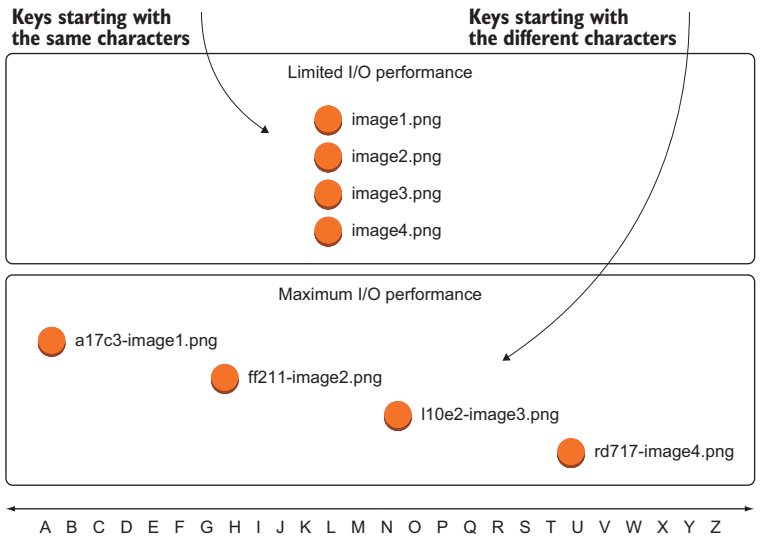

Figure 8.13 To improve I/O performance with S3, don't use keys that start with the same characters.

Using a slash (/) in the key name acts like creating a folder for your object. If you create an object with the key *folder/object.png*, the folder will become visible as *folder* if you're browsing your bucket with a GUI like the Management Console, for example. But technically, the key of the object still is folder/object.png.

Suppose you need to store images that were uploaded by different users. You might come up with the following naming schema for your object keys:

```
upload/images/$ImageId.png
```

$ImageId is a numerical ID that is increased with each new image. A list of your objects might look like this.

```
image1.png
image2.png
image3.png
image4.png
```

The object keys are in alphabetical order, and your maximum throughput with S3 won't be optimal. You can fix this by adding a hash prefix to each object. For example, you can use the MD5 hash of the original key name and prepend it to the key:

```
a17c3-image1.png
ff211-image2.png
l10e2-image3.png
rd717-image4.png
```

This will help distribute your keys across partitions and increase the I/O performance of S3. Knowing about the internals of S3 helps you to optimize your usage.

Summary

- An object consists of a unique identifier, metadata to describe and manage the object, and the content itself. You can save images, documents, executables, or any other content as an object in an object store.
- Amazon S3 is an object store accessible only via HTTP(S). You can upload, manage, and download objects with the CLI, SDKs, or the Management Console.
- Integrating S3 into your applications will help you implement the concept of a stateless server, because you don't have to store objects locally on the server.
- You can define a lifecycle for your objects that will move them from Amazon S3 to Amazon Glacier, a special service for archiving data that you don't need to access frequently. Doing so reduces your cost dramatically.
- S3 is an eventually consistent object store. You have to consider this if you integrate it into your applications and processes, to avoid unpleasant surprises.